DOCUMENT RESUME

ED 192 767 IR 008 838

AUTHOR LaFrance, Jacques E.
TITLE Shall We Teach Structured Programming to Children?
PUB DATE 79
NOTE 11p.

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computer Programs; *Educational Games; Elementary Education; Feasibility Studies; Gifted; *Instructional Innovation; Intermediate Grades; *Programing; Programing Languages

ABSTRACT

To study the effectiveness and feasibility of using structured programming games at the elementary school level, this study presented one and one-half hours of programming instruction to a group of gifted children between the ages of nine and twelve. Using a game called Antfarm and the programming language Pascal, the instruction introduced certain structured programming features such as basic commands, concepts of sequence, iteration, selection and refinement, and the notion of hierarchical structure. Results showed high motivation for the entire group. A few children were beginning to use top-down design, to define their own modules, and to give them names. Two problems noted were that the selection commands were difficult for the children to learn, and technical bugs sometimes caused the program to abort. A 19-item bibliography is attached. (BK)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED192767

Shall We Teach Structured Programming to Children?

by Dr. Jacques E. LaFrance
Department of Mathematical Science
Oral Roberts University
Tulsa, Oklahoma

Abstract

Pre-college programming experience has been observed to be detrimental in many cases to college-level study of computer science. The problem is the lack of understanding of the principles of structured programming. The solution proposed here is the introduction of structured programming games at the elementary school level which will prepare the children to do structured programming later on with languages such as BASIC and FORTRAN that are not designed to promote structured programming. An example of doing this called ANTFARM is presented and the results of using it with one group of children are discussed.

The Problem

More and more students are entering university computer science programs with some form of prior experience with computers, this is typically a high school course in FORTRAN or BASIC or experience with a personal computer. There seems to be no corresponding exposure to structured programming or design, however. The students believe they know a lot because they have written programs in BASIC or FORTRAN, but in reality they know only coding; they understand nothing of structure, top down design, good style, or documentation. It is sometimes difficult to get them to break the resulting bad programming habits. This problem is likely to be compounded by the increasing availability of small personal computers without proper accompanying instruction in good top down programming. An increasing number of young people will likely learn programming by reading their microcomputer BASIC manuals. The learning of coding will thus be encouraged rather than the learning of programming. Yet these students will be

IR008838

8-21

misled into thinking they know a lot because they are able to code some substantial programs. The task of computer science education will be made more difficult by this background experience because the students will have to do more unlearning than is currently the case and sometimes unlearning is more difficult than learning.

The Proposal

A possible solution to the above problem would be to introduce structured programming concepts to children before they are able to begin to use conventional programming language. Other solutions might be to replace BASIC with Pascal as the most common microcomputer language or to have all manufacturers' BASIC manuals based on structured programming. Neither of these alternative solutions seems feasible. The simplicity of BASIC will cause it to continue to be popular, especially with the smallest computer configurations. The large number of programs already coded in some dialect of BASIC will also serve to keep support for BASIC strong. The manufacturers may be more or less willing to include structured programming concepts in their manuals, but the bottom line will always be what enables them to sell their product. The general public will have to become more knowledgeable, discriminating, and particular before the manufacturers will feel any serious pressure to include material on structured programming.

Although the first solution mentioned above is not a shoo-in, it is one that can be developed more easily than either of the others and would be worth investigating. It is not realistic to have all elementary schools begin teaching structured programming, but a few well-published success stories plus the availability of well-written materials for use in schools would help to

promote the idea. A few successful projects at schools in several areas across the country would encourage other educators to do something similar and inspire computer manufacturers to include structured design and programming in their manuals.

Structured programming concepts could easily be introduced to elementary school children if they were presented in a game form at their level of sophistication. Games could be developed for which the control language is inherently structured, and the children would be motivated by the attraction of the game to master the control language as well as possible in order to do more with the game. Because the language would be inherently structured, they would automatically begin to develop structured problem solving skills and structured expression of their ideas. Manuals could be provided that would describe structured design and programming in a way that most teachers could follow and would also describe how to use the games with the children to develop their understanding and skills.

The LOGO system at M.I.T. Artificial Intelligence Laboratory (12) already contributes to these goals. This system was not developed for introducing structured programming concepts but could be easily adapted to that purpose. It is exciting for the children to use (14) and therefore has the necessary motivational characteristics. Several other tools for teaching structured programming could be developed with different appeals, different levels of sophistication, and different organizational structures.

The Experiment

A pilot testing of these ideas was made in April 1979 by the author and Dr. Mary Dee Fosberg of Central State University, Edmond, Oklahoma. To present

the ideas of programming to a group of gifted children, we designed a game called ANTFARM. It was implemented on an IMSAI with a Z80 processor, Digital Microsystems disk, and Infoton 200 CRT using the UCSD Pascal system which we took to the presentation and set up there.

The ANTFARM program consists of drawing an ant facing up in the center of the screen and two rows of food ("@"'s) in the upper left. The goal is to have the ant move over to the food, eat it, and plant new food. The ant looks like:

```
\*/          *1          \ /
 0    or     -0    or    *00-    etc., depending on orientation.
 0            0-         / \
/1\           1\
```

It accepts five basic commands: MOVE, TURN LEFT, TURN RIGHT, EAT, and PLANT. MOVE means move forward one character position in the direction the ant is facing; TURN, LEFT or RIGHT, means a 45° rotation about the first body segment; EAT means to consume whatever is under his head; and PLANT means to plant a new seed with his tail. Seeds start out as "."; 100 time units later they germinate (","); then after 50 more units they sprout (";"). They grow into a stalk ("1") after 50 more units and then into a branching plant ("Y") after 50 more. At the end of 75 more units, a flower ("P") appears and then after 75 more or a total of 400 units, the plant matures into food ("@"). One time unit is the time necessary to execute one operation. Such a growing rate seems to be a satisfactory choice though unrealistically fast.

Each time the screen is updated, the cursor is moved to the home position, and the first line on the screen is erased to await the next input. The user may type any combination of commands on this line. When the return key is pressed the commands are performed. We began by simply showing the children

the effects of individual commands. Then we introduced the concept of sequence by showing that several commands can be listed on the same line.

Since the sequence of commands cannot be longer than one line, the limit on program size is quickly reached. Other commands are then introduced to allow the ant to do more things. From the author's point of view, however, additional commands are given to develop the additional structured programming concepts of iteration, selection, and refinement.

The first additional command illustrates iteration: "DO a command sequence n TIMES," where n is an integer. Instead of "n TIMES," there can be "TO ROW integer" or "TO COLUMN integer." The rows are numbered down the left of the screen and the columns across the top on the second row. This command allows the ant to go through a sequence many times before stopping for the entering of the next command. (The "DO" command can be used in conjunction with the do nothing commands "WAIT" and "REST" to watch plants grow.)

The command to illustrate selection has the form:

IF SEE { PLANT / FOOD / SPROUT / DIRT } { AHEAD / LEFT / RIGHT } command.

If the character space in front (or to front left or front right) contains non-space for "PLANT," "@" for "FOOD," ";" or "1" for "SPROUT," or space for "DIRT," the single command following will be executed; otherwise it will be skipped. This command allows for some interesting conditional tasks such as having the ant look for a row of food to eat or an open furrow in which to plant more food. Although the children were able to use this command it seems to need more development to be stronger. In its present form, it seems too restrictive.

To encourage the development of hierarchical structure and refinement, the command sequences are limited to one line, and a command called "REMEMBER" is given. This command has the form "REMEMBER name command sequence END" which allows the child to give names to command sequences for subsequent use as commands themselves. One popular sequence is

REMEMBER TURNAROUND DO TURN LEFT 4 TIMES END

after which "TURNAROUND" can be used as a command. This subprogram capability is especially helpful when used in connection with the selection command since only one command may be selected or skipped. The commands "QUIT" and "STOP" allow the current subprogram to be ended prematurely (or the ANTFARM program itself to end if it is used at the command line level). This capability allows for powerful search features when used in submodules with selection and iteration commands, such as

REMEMBER MOVETOFOOD DO MOVE IF SEE FOOD AHEAD STOP 70 TIMES END

This will cause the ant to move forward 70 times or until there is food right in front of it, whichever comes first.

One of the tasks the children faced was the planting of a field of four rows of ten plants each. The following is an example of a structured development of a solution:

```
REMEMBER FIELD TWOFURROWS TWOFURROWS END
REMEMBER TWOFURROWS FURROW NEXTRIGHT FURROW NEXTLEFT END
REMEMBER FURROW DO MOVE PLANT 10 TIMES END
REMEMBER NEXTRIGHT DO TURN RIGHT 3 TIMES MOVE TURN RIGHT MOVE MOVE END
REMEMBER NEXTLEFT DO TURN LEFT 3 TIMES MOVE TURN LEFT MOVE MOVE END
```

A field could be planted in the upper right part of the screen by

DO MOVE TO ROW 3 TURN RIGHT TURN RIGHT DO MOVE TO COLUMN 65 FIELD

The final commands available are "FORGET name," which eliminates the named module from the table, and "TELL," which lists on the screen all the "REMEMBER"ed names and their definitions. When the user pushes the return key after the "TELL" display, the screen is cleared and the ant's farm scene is redisplayed as it was prior to the "TELL" display. A future addition will be commands to save and recall the set of defined modules to and from a disk file. These commands will allow a child to build his program from day to day rather than having to type everything over from the beginning at each session.

The Results

We had about 1½ hours with a group of gifted children between the ages of nine and twelve. Interest was universally high throughout the session. The format of the session was that for about 20 to 30 minutes the children were shown some of the hardware aspects of computers; then the ANTFARM was introduced. Since the Infoton has a detached keyboard, the author sat to the side and did all the typing while the children looked at the screen and said what to type. We introduced the ANTFARM features roughly in the same order as in this paper, stopping frequently to ask the children how to get the ant to do some specific task. The children picked up the ideas very quickly and were soon telling us of things they wanted to see the ant do. By the end of the 1½ hours, three or four of the most involved children were beginning to use top-down design to achieve some specific task they wanted the ant to do, defining their own modules and giving them names.

The only weaknesses observed were the selection commands and the bugs. The selection commands were too difficult for the children to use unaided in their present form, primarily because the condition was something happening out

in front of the ant instead of right under him, and the seeing left or right does not correspond well to where the ant is facing when he turns. The bugs occasionally caused the program to abort, losing all the defined modules and all the farm development to that point. Actually the program worked quite well considering it was developed and implemented in three or four man-days and consists of over 400 lines of Pascal code. (This accomplishment in itself is a credit to the value of top-down design, structured programming, and Pascal.)

The Conclusion

The ANTFARM program has demonstrated that it is possible to create tools with high motivational value for children that contain all the concepts of structured programming. Furthermore, the way this program is designed forces the children to use hierarchical or structured development to achieve their goals. Even in one session we were beginning to see some of the children using structured development on their own. It seems reasonable to suggest that continued use of tools such as ANTFARM over an extended period of time could develop these concepts so well in children that they would continue to use top-down design and structured programming even with languages such as BASIC and FORTRAN which are not naturally oriented toward them. We hope that we will have the opportunity to continue to explore these ideas and that others will have similar opportunities. We need long-term data on children using these tools to determine the effects on their future success in programming or computer science.

BIBLIOGRAPHY

1. Banet, Bernard, "Computers and Early Learning," Creative Computing, #4-5 (Sept.-Oct. '78), pp. 90-95.

2. Beyer, Kathleen, and Stuart Milner, "Elementary School Curriculum Task Group," ES[3] Report, September '79, pg. 23-4.

3. Brady, J. M., and R. B. Emanuel, "An Experiment in Teaching Strategic Thinking," Creative Computing, #4-6 (Nov.-Dec. '78), pp. 106-109.

4. Cohen, Harvey A., "Oznaki and Beyond," Proceedings of the National Educational Computing Conference 1979, pp. 170-178.

5. Dahl, O.-J., E. W. Dijkstra, and C. A. R. Hoare, Structured Programming, Academic Press, New York, 1972

6. Hakansson, Joyce, and Leslie Roach, "A Dozen Apples for the Classroom," Creative Computing, #5-9 (Sept. '79), pp. 52-54.

7. Larsen, Sally Greenwood, "Kids and Computers: The Future is Today," Creative Computing, #5-9 (Sept. '79), pp. 58-60.

8. Lieberman, Henry, "The TV Turtle A LOGO Graphics System for Raster Displays," MIT, A. I. Memo 361, (June '76).

9. MECC, "Computer Literacy Objectives from MECC," ACM SIGCUE Bulletin, #13-4 (Oct. '79)

10. McGowan, Clement L., and John R. Kelly, Top-Down Structured Programming Techniques Petrocelli/Charter, New York, 1975.

11. Milner, Stuart D., "An Analysis of Computer Education Needs for K-12 Teachers," Proceedings of the National Educational Computing Conference, 1979, pp. 27-30.

12. Papert, Seymour, "Teaching Children Thinking," MIT, Artificial Intelligence Memo No. 247, (Oct. '71).

13. Papert, Seymour and Cynthia Solomon, "Twenty Things to Do with a Computer," MIT, A. I. Memo 248, (June '71).

14. Papert, Seymour and Harold Abelson, Jeanne Bamberger, Andrea diSessay, Sylvia Weir "Interim Report of the LOGO Project in the Brookline Public Schools: An Assessment and Documentation of a Children's Computer Laboratory," MIT, A. I. Memo No. 484, (June '78).

15. Perlman, Radia, "Using Computer Technology to Provide a Creative Learning Environment for Preschool Children," MIT, A. I. Memo 360, (May '76).

16. Ragsdale, Ronald G., "A Program Package for Introducing the Top-Down Approach to Computer Programming," SIGCSE Bulletin, #11-1 (Feb. '79), pp. 113-117.

17. Solomon, Cynthia J. and Seymour Papert, "A Case Study of a Young Child Doing Turtle Graphics in LOGO," MIT, Artificial Intelligence Memo No. 375, (July '76).

18. Watt, Daniel H., "A Comparison of the Problem-Solving Styles of Two Students Learning LOGO: A Computer Language for Children," Proceedings of the National Educational Computing Conference 1979, pp. 255-260.

19. Weinberg, Gerald M., The Psychology of Computer Programming, Van Nostrand Reinhold, New York, 1971.